

# Unit Testing Strategy

## Philosophy: Testing with Confidence & Clarity

In the Play+ ecosystem, testing is not an afterthought—it's a core part of how we build trust in our work. Just as we aim to design intuitive and adaptive experiences, we also strive to validate them with clarity and confidence. Testing allows us to explore new ideas fearlessly, knowing that well-crafted tests provide a safety net against regressions and subtle bugs.

Our `@playplus/testing-config` package sets the foundation for this approach. It offers a zero-setup, opinionated environment that encourages:

- Frictionless Development : Get started quickly with tools that just work.
- User-Centric Thinking : Write tests that simulate real user behavior, not internals.
- Continuous Integration : Ensure quality is enforced through automated pipelines.

This guide outlines how to write expressive, resilient unit tests for your Play+ React or Angular applications.

## Package Information

The Play+ testing toolchain is provided via the `@playplus/testing-config` package and is a core component of the Golden Path starter kits.

Package / Path	Description
Golden Path (Recommended)	Pre-installed as a devDependency
Uplift Path	<code>npm install --save-dev @playplus/testing-config</code>

## Folder Reference

Play+ follows a test co-location model , meaning test files live directly alongside the code they are testing. This improves discoverability and encourages test creation during development.

File Location	Purpose & Guidelines
src/components/Button.test.tsx	React test file, next to Button.tsx
src/app/features/login.component.spec.ts	Angular test file, next to login.component.ts
reports/coverage/	Git-ignored directory for test coverage reports

## Helper - Pillars Alignment

Pillar	How This Helper Aligns
Intuitive	Primary Pillar: Tests are written from a user's perspective to ensure predictability
Adaptive	Enables safe iteration and refactoring by having a comprehensive test suite
Inclusive	Ensures reliability for all users by testing diverse states and edge cases

## Helper Overview

The playtest toolchain is a set of pre-configured tools, patterns, and CLI helpers designed to abstract the plumbing of modern testing environments.

## What It Automates

- Zero-Setup Environment : Pre-configured Vitest/Jest for React and Jasmine/Karma for Angular.
- Automated Test Scaffolding : CLI helper playtest:gen analyzes components and generates boilerplate.
- Automated Enforcement : Tests run on pre-commit and are required in CI/CD for Golden Path.
- Consistent Mocking : Provides standard mocking for dependencies like apiService .

The goal is to make testing a fast, frictionless, and integral part of development.

## The Play+ Testing Stack

## React

- Test Runner : Vitest (or Jest)
- Testing Library : React Testing Library
- Assertions : expect with matchers from @testing-library/jest-dom

## Angular

- Test Runner : Jasmine and Karma
- Testing Library : Angular TestBed
- Assertions : Jasmine's built-in matchers
- Coverage : Karma Coverage with 80% threshold
- Architecture : Standalone components with testing utilities

## Config Options

Config Variable	Default Value	Description	Recommended Value
coverage.threshold	80	Min % coverage required for CI pass	80
reporters	['default']	Output formats (can include html , json , etc.)	['default', 'html']

## Key Scripts & CLI Commands

Command	What It Does	Example
npm run playwright	Runs full test suite once	npm run playwright
npm run playwright:watch	Starts watch mode for live test re-running	npm run playwright:watch
npm run playwright:gen	Scaffolds a new test file for a given component	npm run playwright:gen src/app/components/UserProfile.ts

## What playwright:gen Generates

## React

- Auto-imports for render and screen
- A describe block and default smoke test
- Placeholders for props and mocks

## **Angular**

- Auto-imports for TestBed and ComponentFixture
- A describe block and default smoke test
- Placeholders for props and mocks
- Accessibility test structure

## **Usage Examples**

### **React**

### **Angular Standalone Components**

### **Core Testing Principles**

### **Follow the AAA Pattern**

- Arrange : Setup component, props, and dependencies
- Act : Simulate user interaction
- Assert : Validate output matches expectations

### **Test Behavior, Not Implementation**

Avoid relying on internal variables:

### **Don't**

### **Do**

### **Mocking Dependencies: Isolating the Unit**

Unit tests should be isolated from external dependencies like APIs.

If apiProxy is detected, playwright:gen will auto-generate mocks.

### **Example (React + Vitest)**

## Example (Angular + Jasmine)

### Testing Utilities

#### PLAY\_TESTING\_IMPORTS

Provides common testing imports for standalone components:

#### PlayTestingUtils

Common testing helpers:

### Additional Info

### Why We Built This

Configuring a modern testing toolchain is complex. It involves selecting and integrating multiple tools (Jest/Vitest, Testing Library, Karma), plugins (for React, Angular, accessibility), and defining hundreds of configuration options. Without a centralized solution, each team would waste time on setup and debates, leading to inconsistencies across projects.

The `@playplus/testing-config` package solves this by providing a single, opinionated, and production-ready configuration. It eliminates boilerplate and configuration drift, ensuring every project starts with and maintains the same high-quality testing standards.

### Best Practices

- Trust the Automation : Let the pre-commit hooks and CI checks do their job.
- Test User Behavior : Focus on what users see and do, not implementation details.
- Use Descriptive Names : Test names should clearly describe the expected behavior.
- Keep Tests Independent : Each test should be able to run in isolation.

### Developer Checklist

- Are you testing from the user's perspective (DOM and behavior, not internals)?
- Are you following the AAA (Arrange, Act, Assert) pattern?
- Are dependencies like APIs properly mocked ?
- Did you cover edge cases and negative flows ?

- Are tests independent , fast, and refactor-proof?
- Did you run `npm run playwright` to validate before pushing?
- Do your tests include accessibility checks ?
- Are you using `PLAY_TESTING_IMPORTS` for consistent setup?

## **Forbidden Patterns**

### **Don't Test Implementation Details**

### **Don't Test Framework Behavior**

### **Don't Use Complex Setup**

## **Required Patterns**

### **Test User Behavior**

### **Use Descriptive Test Names**

### **Group Related Tests**

## **Summary**

Unit testing in Play+ isn't just about preventing bugs—it's about unlocking creative confidence. With robust tooling, opinionated helpers, and automated pipelines, we make testing an asset—not a burden.

- User-Centric : Focus on behaviors and outcomes
- Fully Automated : Enforced via CI to protect quality
- Zero-Setup : Start writing tests immediately
- Isolated & Predictable : Mocks ensure accuracy

Let your tests reflect the experience you're building—not just the code behind it.